Slot-consistent NLG for Task-oriented Dialogue Systems with Iterative Rectification Network

Yangming Li[1,2]*, **Kaisheng Yao**[1]*, **Libo Qin**[2]*, **Wangxiang Che**[2]†, **Xiaolong Li**[1], **Ting Liu**[2]
[1]Ant Financial Services Group, Alibaba Group
[2]Research Center for Social Computing and Information Retrieval,
Harbin Institute of Technology
{pangmao.lym,kaisheng.yao,xl.li}@antfin.com
{lbqin,car,tliu}@ir.hit.edu.cn

Abstract

Data-driven approaches using neural networks have achieved promising performances in natural language generation (NLG). However, neural generators are prone to make mistakes, e.g., neglecting an input slot value and generating a redundant slot value. Prior works refer this to *hallucination phenomenon*. In this paper, we study *slot consistency* for building reliable NLG systems with all slot values of input dialogue act (DA) properly generated in output sentences. We propose Iterative Rectification Network (IRN) for improving general NLG systems to produce both correct and fluent responses. It applies a bootstrapping algorithm to sample training candidates and uses reinforcement learning to incorporate discrete reward related to *slot inconsistency* into training. Comprehensive studies have been conducted on multiple benchmark datasets, showing that the proposed methods have significantly reduced the slot error rate (ERR) for all strong baselines. Human evaluations also have confirmed its effectiveness.

1 Introduction

Natural Language Generation (NLG), as a critical component of task-oriented dialogue systems, converts a meaning representation, i.e., dialogue act (DA), into natural language sentences. Traditional methods (Stent et al., 2004; Konstas and Lapata, 2013; Wong and Mooney, 2007) are mostly pipeline-based, dividing the generation process into sentence planing and surface realization. Despite their robustness, they heavily rely on handcrafted rules and domain-specific knowledge. In addition, the generated sentences of rule-based approaches are rather rigid, without the variance of human language. More recently, neural network based models (Wen et al., 2015a,b; Dušek and Jurčíček, 2016; Tran and Nguyen, 2017a) have attracted much attention. They implicitly learn sentence planning and surface realisation end-to-end with cross entropy objectives. For example, Dušek and Jurčíček (2016) employ an attentive encoder-decoder model, which applies attention mechanism over input slot value pairs.

Input DA	inform(NAME = *pickwick hotel*, PRICERANGE = *moderate*)
Reference	the hotel named *pickwick hotel* is in a *moderate* price range
Missing	this is a *moderate* hotel [NAME]
Misplace	the *pickwick hotel* in *fort mason* is a *moderate* price range [AREA]

Table 1: An exmaple (including mistaken generations) extracted from SF Hotel (Wen et al., 2015b) dataset. Errors are marked in colors (missing, misplaced).

Although neural generators can be trained end-to-end, they suffer from *hallucination phenomenon* (Balakrishnan et al., 2019). Examples in Table 1 show a misplacement error of an unseen slot AREA and a missing error of slot NAME by an end-to-end trained model, when compared against its input DA. Motivated by this observation, in this paper, we define *slot consistency* of NLG systems as all slot values of input DAs shall appear in output sentences without misplacement. We also observe that, for task-oriented dialogue systems, input DAs are mostly with simple logic forms, therefore enabling retrieval-based methods e.g. K-Nearest Neighbour (KNN) to handle the majority of test cases. Furthermore, there exists a discrepancy between the training criterion of cross entropy loss and evaluation metric of slot error rate (ERR), similarly to that observed in neural machine translation (Ranzato et al., 2015). Therefore, it is beneficial to use training methods that integrate the evaluation metrics in their objectives.

*Equal contributions.
† Corresponding author.

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 97–106
July 5 - 10, 2020.

In this paper, we propose Iterative Rectification Network (IRN) to improve *slot consistency* for general NLG systems. IRN consists of a pointer rewriter and an experience replay buffer. Pointer rewriter iteratively rectifies *slot-inconsistent* generations from KNN or data-driven NLG systems. Experience replay buffer of a fixed size collects candidates, which consist of mistaken cases, for training IRN. Leveraging the above observations, we further introduce a retrieval-based bootstrapping to sample *pseudo* mistaken cases as candidates for enriching the training data. To foster consistency between training objective and evaluation metrics, we use REINFORCE (Williams, 1992) to incorporate *slot consistency* and other discrete rewards into training objectives.

Extensive experiments show that, the proposed model, KNN + IRN, significantly outperforms all previous strong approaches. When applying IRN to improve *slot consistency* of prior NLG baselines, we notice large reductions of their slot error rates. Finally, the effectiveness of the proposed methods are further confirmed using BLEU scores, case analysis and human evaluations.

2 Preliminary

2.1 Delexicalization

Inputs to NLG are structured meaning representations, i.e., DA, which consists of an act type and a list of slot value pairs. Each slot value pair represents the type of information and its content while the act type control the style of sentence. To improve generalization capability of DA, delexicalization technique (Wen et al., 2015a,b; Dušek and Jurčíček, 2016; Tran and Nguyen, 2017a) is widely used to replace all values in reference sentence by their corresponding slot in DA, creating pairs of delexicalized input DAs and output templates.

Hence the most important step in NLG is to generate templates correctly given an input DA. However, this step can introduce missing and misplaced slots, because of modeling errors or unaligned training data (Balakrishnan et al., 2019; Nie et al., 2019; Juraska et al., 2018). Lexicalization is followed after a template is generated, replacing slots in template with corresponding values in DA.

2.2 Problem Statement

Formally, we denote a delexicalized input DA as a set $\mathbf{x} = \{x_1, x_2, \cdots, x_N\}$ that consists of an act type and some slots. Universal set $\mathbf{S}$ contains all possible slots. The output template $\mathbf{y} = [y_1, y_2, \cdots, y_M]$ from NLG systems $f(\mathbf{x})$ is a sequence of tokens (words and slots).

We define a slot extraction function g as

$$g(\mathbf{z}) = \{t \mid t \in \mathbf{z}; t \in \mathbf{S}\}. \quad (1)$$

where $\mathbf{z}$ consists of the DA $\mathbf{x}$ and elements of the template $\mathbf{y}$.

A *slot-consistent* NLG system $f(\mathbf{x})$ satisfies the following constraint:

$$g(f(\mathbf{x})) = g(\mathbf{x}). \quad (2)$$

To avoid trivial solutions, we require that $f(\mathbf{x}) \neq \mathbf{x}$.

However, due to the *hallucination phenomenon*, it is possible to miss or misplace slot value in generated templates (Wen et al., 2015a), which is hard to avoid in neural-based approaches.

2.3 KNN-based NLG System

A KNN-based NLG system f^{KNN} is composed of a distant function ρ and a template set $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_Q\}$ which is collected from Q delexicalized sentences in training corpus.

Given input DA $\mathbf{x}$, the distance is defined as

$$\rho(\mathbf{x}, \mathbf{y}_i) = \#(\{s \mid s = t; t \in \mathbf{y}_i; s \in \mathbf{x}\}), \quad (3)$$

where function $\#$ computes the size of a set. During evaluation, system f^{KNN} first ranks the templates in set $\mathbf{Y}$ by distant function ρ and then selects the top k (beam size) templates.

3 Architeture

Figure 1 shows the architecture of Iterative Rectification Network. It consists of two components: a pointer rewriter to produce templates with improved performance metrics and an experience replay buffer to gather and sample training data.

The improvements on *slot consistency* are obtained via an iterative rewriting process. Assume, at iteration k, we have a template $\mathbf{y}^{(k)}$ that is not slot consistent with input DA, i.e., $g(\mathbf{y}^{(k)}) \neq g(\mathbf{x})$. Then, a pointer rewriter iteratively rewrites it as

$$\mathbf{y}^{(k+1)} = \phi^{\text{PR}}(\mathbf{x}, \mathbf{y}^{(k)}). \quad (4)$$

Above recursion ends once $g(\mathbf{y}^{(k)}) = g(\mathbf{x})$ or a certain number of iterations is reached.

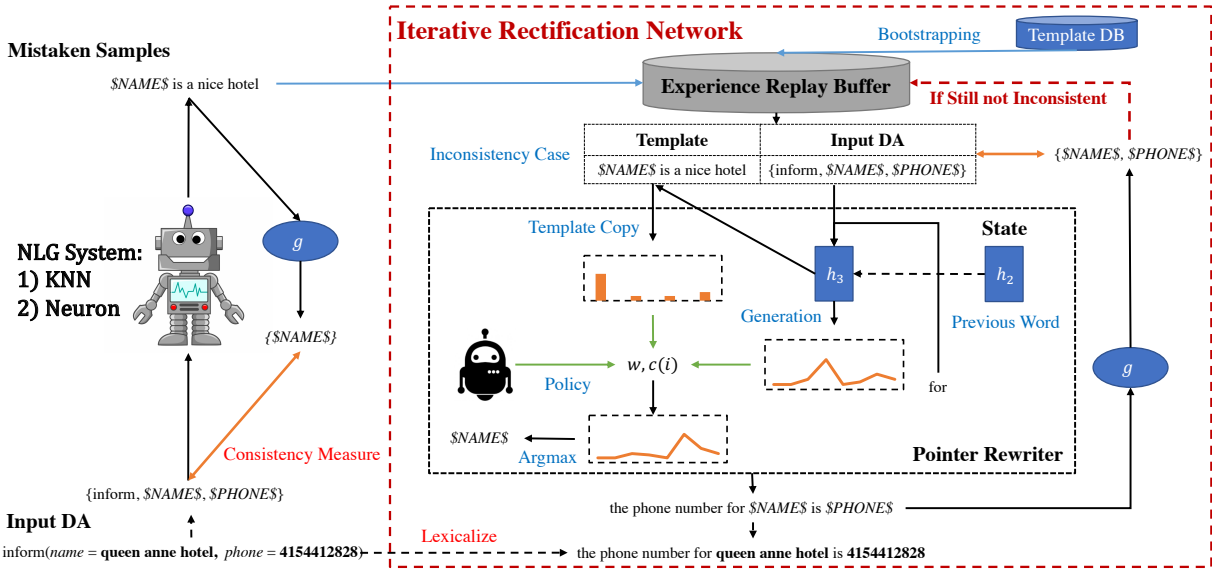

Figure 1: IRN consists of two modules: an experience replay buffer and a pointer rewriter. The experience replay buffer collects mistaken cases from NLG baseline, template and IRN itself (the red dashed arrow) whereas the pointer network outputs templates with improved performance metrics. In each epoch of rectification, IRN obtains samples of cases for training from the buffer and trains a pointer rewriter with metrics such as *slot consistency* using a policy-based reinforcement learning technique. We omit some trivial connections for brevity.

3.1 Pointer Rewriter

The pointer rewriter ϕ^{PR} is trained to iteratively correct the candidate $\mathbf{y}^{(k)}$ given a DA $\mathbf{x}$. This correction operation is conducted time-recurrently. At each position j of rewriting a template, there is a state $\mathbf{h}_j$ to represent the past history of the pointer rewriter and an action a_j to take according to a policy π.

State We use an autoregressive model, in particular LSTM to compute state $\mathbf{h}_j$, given its past state $\mathbf{h}_{j-1}$, input $\mathbf{x}$ and its past output $y^{(k)}_{j-1}$

$$\mathbf{h}_j = \phi^{\text{LSTM}}(\mathbf{h}_{j-1}, [\mathbf{x}; y^{(k)}_{j-1}; \mathbf{c}_j]), \qquad (5)$$

where DA $\mathbf{x}$ is represented by one-hot representation (Wen et al., 2015a,b). $\mathbf{c}_j$ is a context representation over input template $\mathbf{y}^{(k)}$, to be described in Eq. (6). The operation $[;]$ means vector concatenation.

Action For position j in the output template $\mathbf{y}^{(k)}$, its action a_j is in a space consisting of two categories: template copy, $c(i)$, to copy a token from the template $\mathbf{y}^{(k)}$ at i, and word and slot generation, w, to generate a word or a slot at the position. For a length-M input template $\mathbf{y}^{(k)}$, the action a_j is therefore in a set of $\{w, c(1), \cdots, c(M)\}$. The action sequence $\mathbf{a}$ for a length-N output template is $[a_1, \cdots, a_N]$.

Template Copy The model ϕ^{PR} for template copy uses attentive pointer to decide, for position j, what token to copy from the candidate $\mathbf{y}^{(k)}$. Each token $y^{(k)}_i$ in candidate $\mathbf{y}^{(k)}$ is represented using an embedding $\mathbf{y}^{(k)}_i$. For position j in the output template, this model utilizes the above hidden state $\mathbf{h}_j$ and computes attentive weights to all of the tokens in $\mathbf{y}^{(k)}$, with weight to token embedding $\mathbf{y}^{(k)}_i$ as follows:

$$\begin{cases} \phi^{\text{PR}}(\mathbf{h}_j, \mathbf{y}^{(k)}_i) = \mathbf{v}_a^T \sigma(\mathbf{W}_h * \mathbf{h}_j + \mathbf{W}_y * \mathbf{y}^{(k)}_i) \\ \mathbf{p}^{\text{PR}}_{ij} = \text{Softmax}(\phi^{\text{PR}}(\mathbf{h}_j, \mathbf{y}^{(k)}_i)) \\ \mathbf{c}_j = \sum_{1 \le i \le M} \mathbf{p}^{\text{PR}}_{ij} \mathbf{y}_i \end{cases}, \qquad (6)$$

where $\mathbf{v}_a$, $\mathbf{W}_h$, $\mathbf{W}_y$ are learnable parameters.

Word and Slot Generation Another candidate for position j is a word or a slot key from a predefined vocabulary. The action w computes a distribution of words and slot keys below

$$\mathbf{p}^{\text{Vocab}}_j = \text{Softmax}(\mathbf{W}_v * \mathbf{h}_j), \qquad (7)$$

where this distribution is dependent on the state $\mathbf{h}_j$ and matrix $\mathbf{W}_v$ is learnable.

Algorithm 1: Interactive Data Aggregation

Input: template-DB, T;
baseline NLG system, b;
pointer rewriter, ϕ^{PR};
total epoch number, K;
candidate set size, U
Output: ideal pointer rewriter, ϕ^{PR}.

$B, C \leftarrow \{\}, \{\}$
$epoch \leftarrow 0$
for $\mathbf{x}, \mathbf{z} \in T$ **do**
  $\mathbf{y} \leftarrow b(\mathbf{x})$
  if $g(\mathbf{z}) \neq g(\mathbf{y})$ **then**
    $C \leftarrow C + (\mathbf{x}, \mathbf{y}, \mathbf{z})$
  end
end
while $epoch < K$ **do**
  $\Omega \leftarrow$ Bootstrapping$(T, U - |C|)$
  $B \leftarrow C + \Omega$
  Training(ϕ^{PR}, B)
  $C \leftarrow \{\}$
  for $\mathbf{x}, \mathbf{y}, \mathbf{z} \in B$ **do**
    $\hat{\mathbf{y}} \leftarrow \phi^{\mathrm{PR}}(\mathbf{x}, \mathbf{y})$
    if $g(\mathbf{y}) \neq g(\hat{\mathbf{y}})$ **then**
      $C \leftarrow C + (\mathbf{x}, \hat{\mathbf{y}}, \mathbf{z})$
    end
  end
  $epoch \leftarrow epoch + 1$
end

Policy The probabilities for the above actions can be computed as follows

$$\begin{cases} \pi(c(i)|\mathbf{h}_j) = \lambda_j * \mathbf{p}_j^{\mathrm{PR}}(i) \\ \pi(w|\mathbf{h}_j) = (1 - \lambda_j) * \mathbf{p}_j^{\mathrm{Vocab}} \end{cases}, \quad (8)$$

where $\pi(c(i)|\mathbf{h}_j)$ is the probability of copying the i-th token from input template $\mathbf{y}^{(k)}$ to position j. $\pi(w|\mathbf{h}_j)$ is the probability to use words or slot keys predicted from the distribution $\mathbf{p}_j^{\mathrm{Vocab}}$ in Eq. (7). The weight λ_j is a real value between 0 and 1. It is computed from a Sigmoid operation as $\lambda_j = \mathrm{Sigmoid}(\mathbf{v}_h * \mathbf{h}_j)$. With the policy, the pointer rewriter does greedy search to decide whether copying or generating a token.

3.2 Experience Replay Buffer

The experience replay buffer aims at providing training samples for IRN. It has three sources of samples. The first is from off-the-shelf NLG systems. The second is from the pointer rewriter in the last iteration. Both of them are real mistaken samples. They are stored in a case set C in the buffer. These samples are off-policy as the case set C can contain samples from many iterations before. The third source is sampled from a bootstrapping algorithm. They are stored in a set Ω.

Algorithm 2: Bootstrapping via Retrieval

Input: template-DB, T;
total sample number, V;
maximum tolerance (default 2), ϵ.
Output: pseudo sample set, Ω.

$\Omega \leftarrow \{\}$
while $|\Omega| < V$ **do**
  $\mathbf{x}, \mathbf{z} \leftarrow$ RandomSelect(T)
  $Z \leftarrow \{\}$
  for $\hat{\mathbf{x}}, \hat{\mathbf{z}} \in T$ **do**
    $p \leftarrow g(\mathbf{z})$
    $q \leftarrow g(\hat{\mathbf{z}})$
    if $p \neq q \cap |p - q| < \epsilon$ **then**
      $Z \leftarrow Z + (\mathbf{x}, \hat{\mathbf{z}}, \mathbf{z})$
    end
  end
  $\Omega \leftarrow \Omega +$ RandomSelect(Z)
end

Iterative Data Aggregation The replay experiences should be progressive, reflecting improvements in the iterative training of IRN. Therefore, we design an iterative data aggregation algorithm in Algorithm 1. In the algorithm, the experience replay buffer B is defined as a fixed size set of $B = C + \Omega$. For a total epoch number of E, it randomly provides mistaken samples for training pointer rewriter ϕ^{PR} at each epoch. Importantly, both content of C and Ω are varying from each epoch. For C, it initially consists of real mistaken samples from the baseline system (line 3-th to line 8-th). Later on, it's gradually filled by the samples from the IRN (line 14-th to line 19-th). For Ω, its samples reflect a general distribution of training samples from a template database T (line 10-th). Finally, the algorithm aggregates these two groups of mistaken samples (line 11-th) and use them to train the model ϕ^{PR} (line 12-th).

Bootstrapping via Retrieval Relying solely on the real mistaken samples exposes the system to data scarcity problem. It is easy to observe that real samples are heavily biased towards certain slots, and the number of real mistaken samples can be small. To address this problem, we introduce a

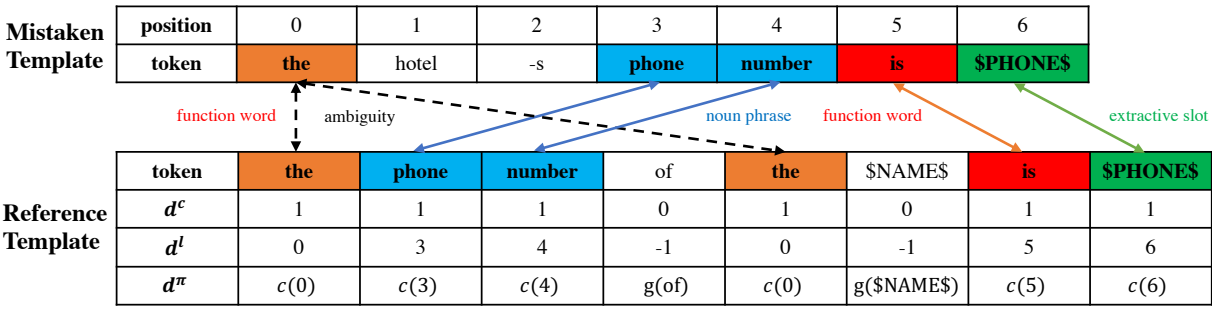

Figure 2: Correcting a candidate given a reference template. d^c, d^l, and d^π are inferred by simple rules.

bootstrapping algorithm, described in Algorithm 2. It uses a template database T, built from delexicalized NLG training corpus and organized by pairs of DA and reference template $(\mathbf{x}, \mathbf{z})$.

At each turn of the algorithm, it first randomly samples (line 3-th) a pair $(\mathbf{x}, \mathbf{z})$, from training template data base of T. Then for every pair $(\hat{\mathbf{x}}, \hat{\mathbf{z}})$ in T, it measures if the pair $(\hat{\mathbf{x}}, \hat{\mathbf{z}})$ is *slot-inconsistent* with respect to $(\mathbf{x}, \mathbf{z})$, and adds the pair that is within a certain distance ϵ (a hyper parameter) to a set Z (line 5-th to 11-th). ϵ is usually set to a small number so that the selected samples are close enough to (x, z). In practice, we set it to 2. Finally, it does a random sampling (line 12-th) on Z and insert its return into the output set Ω. Such bootstrapping process stops when the number of generated samples reaches a certain limit K.

These samples, which we refer them as pseudo samples in the following, represent a wider coverage of training samples than the real mistaken samples. Because they are sampled from general distribution of the templates, some of semantics are not seen in the real mistaken cases. We will demonstrate through experiments that it effectively addresses data scarcity problem.

4 Training with Supervised Learning and Distant Supervision

One key idea behind the proposed IRN model is to conduct distant supervision on the actions of template copy and generation. We diagram its motivation in Figure 2. During training, only candidate $\mathbf{y}$ and its reference $\mathbf{z}$ are given. The exact actions that convert template $\mathbf{y}$ to $\mathbf{z}$ have to be inferred from the two templates. Here we use simple rules for the inference. Firstly, the rules check if reference token $\mathbf{z}_j$ exists in the candidate $\mathbf{y}$. The output is a label d^c consisting of 1s and 0s, representing whether tokens in the reference template are existent/absent in the candidate. Secondly, the rules locate the original position d^l_j in the candidate for each token j in the reference template if $d^c = 1$ and use -1 for $d^c = 0$. Finally, the action label d^π for policy is inferred, with w for $d^l_j = -1$ and $c(i)$ for $d^l_j = i$.

We may use the extracted tags to do supervised learning. The loss to be minimized is as follows

$$J^{\mathrm{SL}} = -\sum_{j=1}^{L} \log \pi(d^\pi_j | \mathbf{h}_j), \tag{9}$$

where L is the length of ground truth. $\pi(d^\pi_j | \mathbf{h}_j)$ computes the likelihood of action d^π_j at position j given state $\mathbf{h}_j$.

However, there are following issues when attempting to utilize the labels produced by distant supervision for training. Firstly, the importance of every token in candidate is different. For example, noun phrase (colored in blue) is critical and should be copied. Function words (colored in red) is of little relevance and can be generated by IRN itself. However, distant supervision treats them the same. Secondly, rule-based matching may cause semantic ambiguity (dashed line colored in black). Lastly, the training criterion of cross entropy is not directly relevant to the evaluation metric using slot error rate. To address these issues, we use reinforcement learning to obtain the optimal actions.

5 Training with Policy-based Reinforcement Learning

In this section, we describe another method to train IRN. We apply policy gradient (Williams, 1992) to optimize models with discrete rewards.

5.1 Rewards

Slot Consistency This reward is related to the correctness of output templates. Given the set of slot-value pairs $g(\mathbf{y})$ from the output template generated by IRN and the set of slot-value pairs $g(\mathbf{x})$ extracted from input DA, the reward is zero when

Model	SF Restaurant		SF Hotel		Laptop		Television	
	BLEU	ERR	BLEU	ERR	BLEU	ERR	BLEU	ERR
HLSTM (Wen et al., 2015a)	0.747	0.74%	0.850	2.67%	0.513	1.10%	0.525	2.50%
SCLSTM (Wen et al., 2015b)	0.753	0.38%	0.848	3.07%	0.512	0.79%	0.527	2.31%
TGen (Dušek and Jurčíček, 2016)	0.751	0.84%	0.853	4.14%	0.515	0.87%	0.521	2.32%
ARoA (Tran and Nguyen, 2017b)	0.776	0.30%	0.892	1.13%	0.522	0.50%	0.539	0.60%
RALSTM (Tran and Nguyen, 2017a)	0.779	0.16%	0.898	0.43%	0.525	0.42%	0.541	0.63%
IRN (+ KNN)	**0.807**	**0.11%**	**0.911**	**0.32%**	**0.537**	**0.29%**	**0.559**	**0.35%**

Table 2: Experiment results on four datasets for all baselines and our model. Meanwhile, the improvements over all prior methods are statistically significant with $p < 0.01$ under t-test.

they are equal; otherwise, it is negative with value set to the cardinality of the difference between the two sets as follows

$$r^{\text{SC}} = -|g(\mathbf{y}) - g(\mathbf{x})|. \tag{10}$$

Language Fluency This reward is related to the naturalness of the realized surface form from a response generation method. Following (Wen et al., 2015a,b), we first train a backward language model on the reference texts from training data. Then, the perplexity (PPL) of the surface form after lexicalization of the output template $\hat{\mathbf{y}}$ is measured using the language model. This PPL is used for the reward for language fluency as follows:

$$r^{\text{LM}} = -\text{PPL}(\mathbf{y}). \tag{11}$$

Distant Supervision We also measure the reward from using distant supervision in Section 4. For a length-N reference template, the reward is given as follows:

$$r^{\text{DS}} = -\sum_{j=1}^{L} \log \pi(d_j^{\pi} | \mathbf{h}_j), \tag{12}$$

where d_j^{π} is the inferred action label.

The final reward for action $\mathbf{a}$ is a weighted sum of the rewards discussed above:

$$r(\mathbf{a}) = \gamma^{\text{SC}} r^{\text{SC}} + \gamma^{\text{LM}} r^{\text{LM}} + \gamma^{\text{DS}} r^{\text{DS}} \tag{13}$$

where $\gamma^{\text{SC}} + \gamma^{\text{LM}} + \gamma^{\text{DS}} = 1$. We set them to equal value in this work. A reward is observed after the last token of the utterance is generated.

5.2 Policy Gradient

We utilize supervised learning in Eq. (9) to initialize our model with the labels extracted from distant supervision. After its convergence, we continuously tune the model using policy gradient described in this section. The policy model in ϕ^{PR} itself generates a sequence of actions $\mathbf{a}$, that are not necessarily the same as d^{π}, and this produces an output template $\mathbf{y}$ to compute slot consistency reward in Eq. (10) and language fluency reward in Eq. (11). With these rewards, the final reward is computed in (13). The gradient to back propagate is estimated using REINFORCE as

$$\nabla J^{\text{RL}}(\theta) = (r(\mathbf{a}) - b) * \sum_{j=1}^{N} \nabla \log \pi(a_j | \mathbf{h}_j), \tag{14}$$

where θ denotes model parameters. $r(\mathbf{a}) - b$ is the advantage function per REINFORCE. b is a baseline. Through experiments, we find that $b = \text{BLEU}(\mathbf{y}, \mathbf{z})$ performs better (Weaver and Tao, 2001) than tricks such as simple averaging of the likelihood $\frac{1}{N} \sum_{j=1}^{N} \log \pi(a_j | \mathbf{h}_j)$.

6 Experiments

6.1 Experiment Setup

We assess the model performances on four NLG datasets of different domains. The SF Hotel and SF Restaurant benchmarks are collected in (Wen et al., 2015a) while Laptop and TV benchmarks are released by (Wen et al., 2016). Each dataset is evaluated with five strong baseline methods, including HLSTM (Wen et al., 2015a), SC-LSTM (Wen et al., 2015b), TGen (Dušek and Jurčíček, 2016), ARoA (Tran and Nguyen, 2017b) and RALSTM (Tran and Nguyen, 2017a). Following these prior works, the evaluation metrics consist of BLEU and slot error rate (ERR), which is computed as

$$ERR = \frac{p + q}{N}, \tag{15}$$

where N is the total number of slots in the DA, and p, q is the number of missing and redundant slots in the generated template, respectively.

Model	SF Restaurant		Television	
	BLEU	ERR	BLEU	ERR
HLSTM (Wen et al., 2015a) w/ **IRN**	0.060 ↑	0.66% ↓	0.040 ↑	2.29% ↓
TGen (Dušek and Jurčíček, 2016) w/ **IRN**	0.002 ↑	0.73% ↓	0.005 ↑	1.99% ↓
RALSTM (Tran and Nguyen, 2017a) w/ **IRN**	0.007 ↑	0.11% ↓	0.004 ↑	0.36% ↓

Table 3: The up and down arrows emphasize the absolutely improved performances contributed by IRN.

Method	Laptop	
	BLEU	SER
IRN (+KNN)	**0.537**	**0.29%**
w/o IRN	0.414	0.88%
w/o reward r^{SC}	0.526	0.75%
w/o reward r^{DS}	0.527	0.66%
w/o reward r^{LM}	0.529	0.49%
w/o baseline $BLEU$	0.531	0.37%
w/o Aggregation	0.515	0.48%
w/o Bootstrapping	0.464	0.83%

Table 4: Ablation study of rewards (upper part) and training data algorithms (lower part).

We follow all baseline performances reported in (Tran and Nguyen, 2017b) and use open source toolkits, RNNLG[1] and Tgen[2] to build NLG systems, HLSTM, SCLSTM and TGen. We re-implement the baselines AROA and RALSTM since their source codes are not available.

6.2 Main Results

We first compare our model, i.e., IRN + KNN with all those strong baselines metioned above. Figure 2 shows that the proposed model significantly outperforms previous baselines on both BLEU score and ERR. Compared with current state-of-the-art model, RALSTM, it achieves reductions of 1.45, 1.38, 1.45 and 1.80 times for SF Restaurant, SF Hotel, Laptop, and Television datasets, respectively. Furthermore, it improves 3.59%, 1.45%, 2.29% and 3.33% of BLEU scores on these datasets, respectively. This improvements of BLEU score can be contributed from language fluency reward r^{LM}.

To verify whether IRN helps improve *slot consistency* of general NLG models, we further equip strong baselines, including HLSTM, TGen and RALSTM, with IRN. We evaluate their performances on SF Restaurant and Television datasets. As shown in Table 3, the methods consistently reduce ERRs and also improve BLEU scores for all baselines on both datasets.

In conclusion, our model, IRN (+ KNN), not only has achieved the state-of-the-art performances but also can contribute to improvements of *slot consistency* for general NLG systems.

[1] https://github.com/shawnwun/RNNLG.
[2] https://github.com/UFAL-DSG/tgen.

Model	Television	
	Informative	Natural
TGen	4.49	3.41
TGen + **IRN**	**4.72**	**3.52**
RALSTM	4.63	4.01
RALSTM + **IRN**	**4.86**	**4.07**

Table 5: Real user trial for generation quality evaluation on both informativeness and naturalness.

6.3 Ablation Study

We perform a set of ablation experiments on the SCLSTM+IRN models on Laptop dataset to understand the relative contribution of data aggregation algorithms in Sec. 3.2 and rewards in Sec. 5.1.

6.3.1 Effect of Reward Designs

The results in Table 4 show that removal of *slot consistency* reward r^{SC} or distant supervision reward r^{DS} from advantage function dramatically degrades SER performance. Language fluency related information from baseline $BLEU$ and reward r^{LM} also have positive impact on BLEU and SER, though they are smaller than using r^{SC} or r^{DS}.

6.3.2 Effect of Data Algorithms

Using only candidates from baselines degrades performance to approximately that of the baseline SCLSTM. This shows that incorporating candidates from IRN is important. The model without bootstrapping, even including candidates from IRN, has worse performance than SCLSTM in Table 3. This shows that bootstrapping to include generic samples from templates database is critical.

6.4 Human Evaluation

We evaluate IRN and some strong baselines on TV dataset. Given an input DAs, we ask human eval-

Input DA	recommend(NAME = **crios 93**, FAMILY = **l1**, AUDIO= **nicam stereo**, SIZE = **large**)
Reference Text	the **large crios 93** television in the **l1** family features **nicam stereo**
Mistaken Generation	the $NAME$ is in $FAMILY$ with $SIZE$ screen and cost about $PRICE$ [AUDIO, PRICE]
1-st IRN Revision	the $NAME$ is a nice television in $FAMILY$ with a $SIZE$ screen [AUDIO]
2-st IRN Revision	the $NAME$ is very nice in $FAMILY$ with a $SIZE$ screen size [AUDIO]
3-st IRN Revision	the $NAME$ is very nice in the $FAMILY$ family with a $SIZE$ screen size and $AUDIO$
Lexicalized Form	the **crios 93** is very nice in the **l1** family with a **large** screen size and **nicam stereo**

Table 6: A DA from Television dataset and a candidate from HLSTM on the DA. The output template from each iteration of IRN. Slot errors are marked in colors (missing, misplaced).

uator to score generated surface realizations from our model and other baselines in terms of informativeness and naturalness. Here informativeness measures whether output utterance contains all the information specified in the DA without insertion of extra slots or missing an input slot. The naturalness is defined as whether it mimics a response from a human (both ratings are out of 5).

Table 5 shows that RALSTM + IRN outperforms RALSTM notably in informativeness relatively by 4.97%, from 4.63 to 4.86. In terms of naturalness, the improvement is from 4.01 to 4.07, relative by 1.50%. Meanwhile, IRN helps to improve the performances of TGen by 5.12% on informativeness and 3.23% on naturalness.

These subjective assessments are consistent to the observations in Table 3, which both have verified the effectiveness of proposed method.

6.5 Case Study

Table 6 presents a sample on TV dataset and shows a progress made by IRN. Given an input DA, the baseline HLSTM outputs in the third row a template that misses slot $AUDIO$ but inserts slot $PRICE$. The output template from the first iteration of IRN has a removal of the inserted $PRICE$ slot. The second iteration has improved language fluency but no progress in slot-inconsistency. The third iteration achieves slot consistency, after which a natural language, though slightly different from the reference text, is generated via lexicalization.

7 Related Work

Conventional approaches for solving NLG task are mostly pipeline-based, dividing it into sentence planning and surface realisation (Dethlefs et al., 2013; Stent et al., 2004; Walker et al., 2002). Oh and Rudnicky (2000) introduce a class-based n-gram language model and a rule-based reranker. Ratnaparkhi (2002) address the limitations of n-gram language models by using more sophisticated syntactic dependency trees. Mairesse and Young (2014) employ a phrase-based generator that learn from a semantically aligned corpus. Despite their robustness, these models are costly to create and maintain as they heavily rely on handcrafted rules.

Recent works (Wen et al., 2015b; Dušek and Jurčíček, 2016; Tran and Nguyen, 2017a) build data-driven models based on end-to-end learning. Wen et al. (2015a) combine two recurrent neural network (RNN) based models with a CNN reranker to generate required utterances. Wen et al. (2015b) introduce a novel SC-LSTM with an additional reading cell to jointly learn gating mechanism and language model. Dušek and Jurčíček (2016) present an attentive neural generator to apply attention mechanism over input DA. Tran and Nguyen (2017b,a) employ a refiner component to select and aggregate the semantic elements produced by the encoder. More recently, domain adaptation (Wen et al., 2016) and unsupervised learning (Bahuleyan et al., 2018) for NLG also receive much attention.

We are also inspired by the post-edit paradigm (Xia et al., 2017), which uses a second-pass decoder to improve the translation quality.

A recent method in (Wu et al., 2019) defines an auxiliary loss that checks if the object words exist in the expected system response of a task-oriented dialogue system. It would be interesting to apply this auxiliary loss in the proposed method. On the other hand, the REINFORCE (Williams, 1992) algorithm applied in this paper is more general than (Wu et al., 2019) to incorporate other metrics, such as BLEU.

Nevertheless, end-to-end neural-based generators suffer from hallucination problem and are hard to avoid generating *slot-inconsistent* utterance (Balakrishnan et al., 2019). Balakrishnan et al. (2019) attempts to alleviate this issue by employing a tree-structured meaning representation and constrained

decoding technique. However, the tree-shaped structure requires additional human annotation.

8 Conclusion

We have proposed Iterative Rectification Network (IRN) to improve *slot consistency* of general NLG systems. In this method, a retrieval-based bootstrapping is introduced to sample pseudo mistaken cases from training corpus to enrich the original training data. We also employ policy-based reinforcement learning to enable training the models with discrete rewards that are consistent to evaluation metrics. Extensive experiments show that the proposed model significantly outperforms previous methods. These improvements include both of correctness measured with slot error rates and naturalness measured with BLEU scores. Human evaluation and case study also confirm the effectiveness of the proposed method.

Acknowledgement

This work was supported by the National Natural Science Foundation of China (NSFC) via grant 61976072, 61632011 and 61772153. This work was done while the first author did internship at Ant Financial. We thank anonymous reviewers for valuable suggestions.

References

Hareesh Bahuleyan, Lili Mou, Kartik Vamaraju, Hao Zhou, and Olga Vechtomova. 2018. Probabilistic natural language generation with wasserstein autoencoders. *arXiv preprint arXiv:1806.08462*.

Anusha Balakrishnan, Jinfeng Rao, Kartikeya Upasani, Michael White, and Rajen Subba. 2019. Constrained decoding for neural NLG from compositional representations in task-oriented dialogue. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. To appear.

Nina Dethlefs, Helen Hastie, Heriberto Cuayáhuitl, and Oliver Lemon. 2013. Conditional random fields for responsive surface realisation using global features. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1254–1263, Sofia, Bulgaria. Association for Computational Linguistics.

Ondřej Dušek and Filip Jurčíček. 2016. Sequence-to-sequence generation for spoken dialogue via deep syntax trees and strings. *arXiv preprint arXiv:1606.05491*.

Juraj Juraska, Panagiotis Karagiannis, Kevin K. Bowden, and Marilyn A. Walker. 2018. A deep ensemble model with slot alignment for sequence-to-sequence natural language generation. In *Proceedings of NAACL-HLT*, pages 152–162.

Ioannis Konstas and Mirella Lapata. 2013. A global model for concept-to-text generation. *Journal of Artificial Intelligence Research*, 48:305–346.

François Mairesse and Steve Young. 2014. Stochastic language generation in dialogue using factored language models. *Computational Linguistics*, 40(4):763–799.

Feng Nie, Jin-Ge Yao, Jinpeng Wang, Rong Pan1, and Chin-Yew Lin. 2019. A simple recipe towards reducing hallucination in neural surface realisation. In *Proceedings of the 57nd Annual Meeting of the Association for Computational Linguistics (ACL-19)*, page 2673–2679, Florence, Italy.

Alice H Oh and Alexander I Rudnicky. 2000. Stochastic language generation for spoken dialogue systems. In *ANLP-NAACL 2000 Workshop: Conversational Systems*.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*.

Adwait Ratnaparkhi. 2002. Trainable approaches to surface natural language generation and their application to conversational dialog systems. *Computer Speech & Language*, 16(3-4):435–455.

Amanda Stent, Rashmi Prasad, and Marilyn Walker. 2004. Trainable sentence planning for complex information presentations in spoken dialog systems. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 79–86, Barcelona, Spain.

Van-Khanh Tran and Le-Minh Nguyen. 2017a. Natural language generation for spoken dialogue system using rnn encoder-decoder networks. *arXiv preprint arXiv:1706.00139*.

Van-Khanh Tran and Le-Minh Nguyen. 2017b. Neural-based natural language generation in dialogue using rnn encoder-decoder with semantic aggregation. *arXiv preprint arXiv:1706.06714*.

Marilyn A Walker, Owen C Rambow, and Monica Rogati. 2002. Training a sentence planner for spoken dialogue using boosting. *Computer Speech & Language*, 16(3-4):409–433.

Lex Weaver and Nigel Tao. 2001. The optimal reward baseline for gradient-based reinforcement learning. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 538–545. Morgan Kaufmann Publishers Inc.

Tsung-Hsien Wen, Milica Gasic, Dongho Kim, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. 2015a. Stochastic language generation in dialogue using recurrent neural networks with convolutional sentence reranking. *arXiv preprint arXiv:1508.01755*.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Lina M Rojas-Barahona, Pei-Hao Su, David Vandyke, and Steve Young. 2016. Multi-domain neural network language generation for spoken dialogue systems. *arXiv preprint arXiv:1603.01232*.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. 2015b. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745*.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.

Yuk Wah Wong and Raymond Mooney. 2007. Generation by inverting a semantic parser that uses statistical machine translation. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 172–179.

Chien-Sheng Wu, Richard Socher, and Caiming Xiong. 2019. Global-to-local memory pointer networks for task-oriented dialogue. In *International Conference on Learning Representations*.

Yingce Xia, Fei Tian, Lijun Wu, Jianxin Lin, Tao Qin, Nenghai Yu, and Tie-Yan Liu. 2017. Deliberation networks: Sequence generation beyond one-pass decoding. In *Advances in Neural Information Processing Systems*, pages 1784–1794.